Planning with Prioritized Goals

Robert Feldmann
Swiss Federal Institute of Technology
Institute of Astronomy
Schafmattstr. 16
CH-8093 Zürich, Switzerland
feldmann@phys.ethz.ch

Gerhard Brewka
University of Leipzig
Dept. of Computer Science
Augustusplatz 10-11
D-04109 Leipzig, Germany
brewka@informatik.uni-leipzig.de

Sandro Wenzel
University of Leipzig
Institute of Theoretical Physics
Augustusplatz 10-11
D-04109 Leipzig, Germany
wenzel@itp.uni-leipzig.de

Abstract

In this paper we present an approach to planning with prioritized goal states. To describe the preference ordering on goal states, we make use of ranked knowledge bases which induce a partial preference ordering on plans. We show how an optimal plan can be computed by assigning an integer value to each state in an appropriate manner. We also show how plan optimality can be tested in a similar fashion. Our implementation is based on `Metric-FF`, one of the fastest existing planning systems. A first empirical evaluation shows very promising results.

Introduction

Classical planning distinguishes between goal states and non-goal states. If there is no plan leading to one of the goal states, then classical planning simply fails. Agents in realistic environments cannot simply refrain from acting if not all of their goals are achievable. Obviously, in such situations the rational thing for an agent to do is trying to achieve the goals in the best possible way. This requires information about the relative quality of reachable states. In other words, what is needed is a preference relation on states. The planning task then consists in finding an optimal plan, that is, a plan leading to a state which is optimal according to the given preference relation on states.

In general, the state space in planning is very large – exponential in the number of atoms used to describe a domain. For this reason describing the preference relation on states explicitly, e.g. by enumerating all pairs in the relation, is out of the question. What is needed is a language which allows the preferences to be described concisely.

In this paper we will use logical formulae to describe the preference relation. More precisely, a ranked knowledge base consisting of formulae representing goals together with a total preorder describing their relative importance is used. Ranked knowledge bases were already proposed in (Brewka 1989) and have proven useful, for instance in reasoning with prioritized defaults.

A ranked goal base induces a partial preorder on plans describing the quality of plans in a purely qualitative fashion.

Our focus on qualitative preferences is motivated by the fact that users are often reluctant to specify numbers when they are asked to describe their preferences. Qualitative preferences are much easier to elicit and sufficient for many applications.

In this paper we address two related, yet different questions in the context of planning with qualitative goal preferences:

1. Given a planning problem with ranked goals, how to compute optimal plans?
2. Given a planning problem with ranked goals and a plan P, how to test whether P is optimal?

Considering the first question we focus on the task of computing a single optimal plan. One of our design principles was to make the computation as independent as possible of the particular planning algorithm. This allows us to use existing planning technology and to benefit from further developments in planning. It is in contrast with approaches like (Brafman & Chernyavsky 2005) which rely on a particular type of planners transforming the original planning problem into a constraint problem (see the discussion of related work at the end of the paper for more on this issue).

Our approach is based on the observation that a state is optimal with respect to the original partial preorder if it is optimal with respect to a total preorder suitably extending the original order. The total preorder can conveniently be expressed using an integer value which is assigned to each state.

Our planning algorithm uses this value as a lower bound in a *generate and improve* method: we start with the computation of a plan for an arbitrary goal state and compute its value. We then iteratively call a classical planner using the value of the most recently found reachable state as lower bound. This way a strictly improving sequence of plans is generated which is guaranteed to converge to an optimal plan. We have implemented our planning algorithm using the `Metric-FF` planner, one of the fastest planning systems available to date.

The second question, testing plan optimality, is also based on a numerical reformulation of the original problem. Given a plan P terminating in state s, we assign an integer value val_P to each state such that P is optimal iff there is no plan P' terminating in s' such that $val_P(s) < val_P(s')$.

The rest of this paper is organized as follows. In the next section we define what we mean by a plan optimization problem (also called partial satisfaction problem in (van den Briel *et al.* 2004)), and by an optimal plan. We then introduce our preference description language using ranked knowledge bases. Subsequently, we present our approach of solving partial satisfaction problems. To test our ideas, we show a realization of our preference languages extending PDDL. We briefly comment on further features of our description languages, e.g. the possibility of mixing qualitative and quantitative preference information. In a further section we describe our implementation and evaluation results. After a discussion of how plan optimality can be tested, the last section describes related work and concludes.

Plan optimization

We first recall the definition of a classical planning problem, following the textbook (Ghallab, Nau, & Traverso 2004). A classical planning problem (CPP) is a 5-tuple $\Gamma = (S, A, \gamma, s_I, S_G)$ consisting of a set[1] of states S, a set of actions A, a transition function $\gamma : S \times A \to S$, an initial state s_I and a description of goal states $S_G \subseteq S$. The transition function can be naturally extended to a function γ^* on action sequences:

$$\gamma^*(s, \langle a_1, \ldots, a_{n-1}, a_n\rangle) := \gamma(\gamma^*(s, \langle a_1, \ldots, a_{n-1}\rangle), a_n)$$

and $\gamma^*(s, \langle\rangle) := s$. We say a state s' is reachable from state s if there exists a finite sequence of actions $A = \langle a_1, \ldots, a_n\rangle$ such that $\gamma^*(s, A) = s'$. A state s' is called solvable if it is reachable from the initial state s_I. A classical planning problem is solvable if there is a solvable state $s_G \in S_G$. A corresponding finite sequence of actions $A = \langle a_1, \ldots, a_n\rangle$ is called solution or plan of length n.

The following small extension leads to the definition of a partial satisfaction problem (PSP). A PSP is a 6-tuple $(S, A, \gamma, s_I, S_G, \succeq)$ where the relation $\succeq\, \subseteq S_G \times S_G$ is a preorder, i.e. a reflexive and transitive relation. Intuitively, $s \succeq s'$ expresses that state s is at least as preferred as state s'. As usual the preorder induces a strict partial order (denoting strict preference) as follows: $s \succ s'$ iff $s \succeq s'$ and $s' \not\succeq s$. The other elements in the 6-tuple are understood as above.

We will use the terms "partial satisfaction problem" and "plan optimization problem" synonymously. The former reflects the fact that the goal states in S_G no longer represent completely satisfactory goal states. Some of them, namely those for which strictly better goal states exist, satisfy the agent's goals only partially. The latter term focuses on the fact that taking partially satisfactory goal states into account also requires information about their respective quality, and that this quality needs to be optimized.

A PSP is solvable if there exists a solvable state $s_G \in S_G$. Whenever a PSP is solvable, there is also an optimal solvable state, i.e. a solvable state s such that for no solvable state $s' \in S_G$ we have $s' \succeq s$ and $s \not\succeq s'$. An optimal plan, also called a solution to the PSP, is a plan leading to an optimal state, that is a sequence $A = \langle a_1, \ldots, a_n\rangle$ such that $\gamma^*(s_I, A)$ is optimal among the solvable states.

In the rest of this paper we assume that the states in a planning problem are represented as subsets of a finite set of atoms $\mathcal{A}$. Intuitively, $s = \{a_1, \ldots, a_k\}$ denotes the state in which the atoms $\{a_1, \ldots, a_k\}$ are true and all other atoms are false. From the point of view of logic, states thus correspond to propositional models, and we will use the usual logical satisfaction symbol $\models$ to denote satisfaction of a formula in a state with the obvious meaning.

[1]For practical purposes all relevant sets are assumed to be finite.

Describing preferences on states

In this section we want to address the question how to represent the preference relation $\succeq$ on goal states in a PSP. The number of states is exponential in the number of atoms. Therefore, an enumeration of the pairs in the preference relation is unfeasible and we need a concise representation of $\succeq$.

We will use logical formulae to represent preferences among states. In the simplest case we can use a single formula f and define $s_1 \succeq s_2$ iff $s_2 \models f$ implies $s_1 \models f$. In many cases, however, a single formula will not be sufficient and we may want to distinguish important from less important formulae.

For this reason we will use ranked knowledge bases (RKBs) (Brewka 1989; Benferhat *et al.* 1993; Pearl 1990; Goldszmidt & Pearl 1991), sometimes also called stratified knowledge bases, to describe $\succeq$ in this paper. Such knowledge bases have proven fruitful in a number of approaches. As discussed in (Brewka 2004), an RKB alone is not sufficient to determine the preference relation on states, even if all formulae are interpreted as goals. In addition, we need a preference strategy which tells us how to use the RKB for this purpose. Several such strategies together with a preference description language allowing to combine them are presented in (Brewka 2004). For the purposes of this paper we will restrict our discussion to one particular such strategy.

A ranked knowledge base (RKB) is a finite set F of propositional formulae together with a total preorder $\geq$ on F. An RKB can be conveniently represented as a sequence $(F_1, \ldots, F_n)$ of sets of formulae such that $f \geq f'$ iff for some j: $f \in F_j$ and for no $i > j$: $f' \in F_i$.

Intuitively, the formulae in F_n represent the most important goals, those in F_{n-1} the most important ones among the other goals etc. The preorder on states induced by an RKB is formally defined as follows:

Definition 1 *Let $K = (F_1, \ldots, F_n)$ be an RKB, S a set of states. For $s \in S$, $j \in \{1, \ldots, n\}$ let*

$$F_j(s) := \{f \in F_j \mid s \models f\}.$$

The preorder on S induced by K, denoted $\succeq_K$, is defined as

$s_1 \succeq_K s_2$ *iff* $F_j(s_1) = F_j(s_2)$ *for all* $j \in \{1, \ldots, n\}$*, or there is a* j *such that* $F_j(s_1) \supset F_j(s_2)$*, and for all* $i > j$: $F_i(s_1) = F_i(s_2)$*.*

According to this definition, a state s is considered strictly better than s' if – starting from level n and proceeding stepwise to the less preferred levels – at the first level where the satisfied formulae do not coincide s satisfies a proper superset of formulae.

It is not difficult to see that $\succeq_K$ is indeed a preorder and not necessarily a total one. Since we assume that the preference relation on states is given by a ranked knowledge base K as just described, we will from now on consider a PSP to be a tuple of the form $\Gamma = (S, A, \gamma, s_I, S_G, K)$.

Computing an optimal plan

PSPs are considered to be harder than classical planning problems since finding a plan for an arbitrary element of S_G is not enough. An optimal solvable element w.r.t. the given preference relation needs to be found. Unfortunately, an optimal solvable element is usually not known a priori. Instead, we have to search the set of goals for a maximum. As a naive approach we could randomly take an element s of S_G and try to find a plan. If one is found, all elements which are strictly smaller than s are marked as inferior and will not be considered further. If no solution is found, the element s and possibly certain other elements are marked unsolvable. The state s, the inferior and the unsolvable elements are removed from S_G and the iteration is repeated. Although this algorithm is somewhat better than a full exhaustive search, it does not scale well to large spaces S_G. Very often a search for a plan for an unsolvable goal state is made, resulting in high performance cost for search failures.

The approach adopted in this paper is based on the fact that a state is optimal w.r.t. the preorder $\succeq$ whenever it is optimal w.r.t. a suitable total order extending $\succeq$ which we will call its linearization.

Let R be a preorder over some domain M of elements. In our case the domain will be the set S_G of goal states. The derived strict preorder $R^>$ and the derived equality preorder $R^=$ are then defined as follows:

$$(m, m') \in R^> \text{ iff } (m, m') \in R \text{ and } (m', m) \notin R \quad (1)$$

$$(m, m') \in R^= \text{ iff } (m, m') \in R \text{ and } (m', m) \in R. \quad (2)$$

The linearized preorder R_{lin} over the same domain M, the corresponding derived strict and equality preorders should satisfy the following requirements, for reasons we will see below:

$$R_{\text{lin}} \supseteq R, \quad \text{(3a)}$$

$$R^=_{\text{lin}} \supseteq R^=, \quad \text{(3b)}$$

$$R^>_{\text{lin}} \supseteq R^>. \quad \text{(3c)}$$

The inclusion (3a) describes the natural assumption that two elements are in relation R_{lin} if there are already in relation R. The last inclusion (3c) prevents a weakening of the relation R by linearization which is too strong. The three requirements are not independent. Inclusion (3a) implies (3b). (3b) and (3c) imply (3a).

Proposition 2 *Let R be a preorder over the finite domain M, then there exists a linear preorder R_{lin} which obeys* (3a)-(3c).

The proof of the proposition (which is omitted due to space restrictions, see (Feldmann 2005) for further details) gives an explicit construction of R_{lin}.

Proposition 3 *Let m^* be an optimal (solvable) element with respect to R_{lin}, then m^* is also an optimal (solvable) element with respect to R.*

Proof: Let m be optimal with respect to R_{lin}. Assume there is m' such that $(m', m) \in R^>$, i.e. m' is strictly better than m with respect to R. But then (3c) implies $(m', m) \in R^>_{lin}$ which is a contradiction to the optimality of m with respect to R_{lin}. □

Linearizations of partial preorders can be conveniently represented using integers. We will now define for each RKB K a valuation function[2] val_K which assigns an integer value to each goal state such that $s \succ_K s'$ implies $val_K(s) > val_K(s')$.

Definition 4 *Let $K = (F_1, \ldots, F_n)$ be an RKB, s a goal state. Let $maxval_0 := 0$ and for each j $(1 \leq j \leq n)$:*

$$\begin{aligned} val_j &:= maxval_{j-1} + 1 \\ maxval_j &:= |F_j| \times val_j + maxval_{j-1} \end{aligned}$$

The K-value of s,$val_K(s)$ is defined as

$$val_K(s) := \sum_{i=1}^{n} |\{g \in F_i \mid s \models g\}| \times val_i.$$

To see how this definition works consider the RKB

$$(\{a\}, \{b, c\}, \{d\}).$$

We have $val_1 = 1, val_2 = 2$ and $val_3 = 6$. The intuition is that satisfying a single goal of higher level leads to a higher value than an arbitrary number of goals of lower levels. Let $s = \{d\}$ and $s' = \{a, b, c\}$. As intended, s gets a higher value (6) than s' (5) reflecting the fact that $s \succ_K s'$.

Proposition 5 *The order on goal states R_{val_K} induced by val_K by defining $(s, s') \in R_{val_K}$ iff $val_K(s) \geq val_K(s')$ is a linearization of $\succeq_K$.*

Proof: Since val_K maps states to integers, R_{val_K} is obviously linear. We have to show $\succeq^=_K \subseteq R^=_{val_K}$ (3b) and $\succeq^>_K \subseteq R^>_{val_K}$ (3c). These two properties imply (3a). (3b) follows from the fact that $(s, s') \in \succeq^=_K$ iff s and s' satisfy exactly the same goals, in which case $val_K(s) = val_K(s')$. (3c) follows from the fact that whenever $(s, s') \in \succeq^>_K$ there must be a level k such that s satisfies a proper superset of level k goals satisfied by s', and the same goals as s' in all levels with higher index. By construction, whatever the satisfied goals of levels with lower index are, $val_K(s) > val_K(s')$. □

We can thus find an optimal solution of a PSP by using the valuation function val_K. More precisely, we use a function ϕ_K on arbitrary states defined as $\phi_K(s) = val_K(s) + 1$ if $s \in S_G$, $\phi_K(s) = 0$ otherwise. The addition of 1 makes sure that each goal state has a non-zero value. This is useful for the algorithm presented below.

Let `solve` be a complete and correct planning algorithm for classical planning problems which returns

- a tuple (s, π) consisting of a solvable goal state s and a corresponding plan π if the classical planning problem is solvable.

[2]We also use the term metric for valuation functions.

Algorithm 1 Solving PSP $\Gamma = (S, A, \gamma, s_I, S_G, K)$

$n := 0$;
$goals := S_G$;
repeat
  $goals := \{s \mid s \in S_G, \phi_K(s) > n\}$;
  $(s, \pi) :=$ `solve`$(S, A, \gamma, s_I, goals)$;
  $n := \phi_K(s)$;
  print $\pi\ s\ n$
until $s \notin goals$

- the initial state (which is a non-goal state in this case) and an empty plan if the planning problem is not solvable.

Furthermore, let Γ be a given PSP. Algorithm 1 will compute an optimal solution for Γ if there exists at least one solution. The algorithm works as follows:

1. It applies the classical planner to the problem (S, A, γ, s_I, S_G), because for $n = 0$
$$S'_G = \{s \mid s \in S_G, \phi(s) > n\}$$
is just the set of solvable goal states. If this is unsolvable then Γ is unsolvable and the algorithm stops.
2. The algorithm tries to increase the quality of a solution of the classical planning problem by excluding all goal states which are not better than the current best solvable goal state s_E. Then it applies `solve` to the more demanding problem
$$(S, A, \gamma, s_I, S'_G = \{s \mid s \in S_G, \phi(s) > \phi(s_E)\}).$$
3. In each iteration step the quality of the found solvable goal state increases. Formally the algorithm terminates if it finds no better solvable goal state. The best solvable goal state obtained and its corresponding plan (assuming completeness and soundness of the classical planner `solve`) constitute a solution of the PSP.

The following aspects seem important for the evaluation of Algorithm 1 :

- Goal states are described as sets of atomic formulae. The number of goal states can be exponential in the number of atoms. Searching naively for a optimal solvable solution with respect to a given preference relation requires checking a vast amount of goal states. Using a linearized version of the preference relation, this search space is usually much smaller.
- In principle, any classical planner can be applied within Algorithm 1 as long as it supports the restriction of the goal set using a numerical comparison. In the following section we will show how this can be done in `PDDL2.1`. We tested this transformation using the publicly available planner `Metric-FF` (Hoffmann 2003).
- The planner will often find better solvable goal states than required by the limit n in each iteration. This accomplishes a fast determination of solvable goal states with high preferences. In the special case where s_I is already an optimal solvable goal state, the algorithm quickly confirms this solution.
- Searching for a plan unsuccessfully takes a lot of time. In Algorithm 1 such a search can happen only once, after an optimal solvable goal state was found.
- Algorithm 1 is presented as an external function which uses the underlying planner as black-box. In doing so, we gained independence of the particular planning method at the expense of performance. However, depending on the planner at hand, it should pose no serious problem to implement our proposed algorithm directly into the planner. In this way the performance can be increased since it is now possible to reuse information from previous iterations.
- The algorithm is sound and complete if the underlying classical planner has the same properties. It can also be used as an anytime-algorithm which prints successively the best solvable goal state found so far.

Expressing PSP in PDDL based languages

In 1998 `PDDL` (Ghallab *et al.* 1998), a new representation language for planning problems extending `ADL` (Pednault 1989) and `STRIPS` (Fikes & Nilsson 1971), has been published. This language has become a de facto standard due to the increasing needs for describing more realistic and therefore more complex planning problems.

A recent version, `PDDL2.1` (Fox & Long 2003), also supports numerical constructs. More precisely, in addition to atomic formulae (as in `STRIPS`) `PDDL2.1` also supports *goal descriptions* (denoted as `<GD>`) which consist of complex logical formulae, numerical variables and numerical comparisons. The exact syntax and semantics of goal descriptions can be found in the definition of the `PDDL2.1` language (Fox & Long 2003). Very briefly, `PDDL2.1` states can be represented as tuples $s = (f, \nu)$ consisting of a set f of atomic formulae and a $k-$tuple ν of rational numbers. The ith component of ν contains the value of the ith numerical variable v_i. An atomic formula is true in a state s if it is contained in f. A numerical comparison holds if the values of the variables in ν are consistent with the comparison. Using atomic formulae and numerical comparisons as their building blocks, it is now possible to define when goal descriptions are satisfied in a state.

It should be noted that most of today's classical planners support a major fraction of `PDDL2.1`, or go even beyond. In order to benefit from further improvements in this field, we decided to develop extensions of `PDDL2.1` to express preferences and partial satisfaction problems. More precisely, we introduce two such extensions. The first one, called `PDDL2.1`^{q}, allows us to represent purely qualitative goal preferences based on RKBs. Its syntax is very simple. The induced preorder has to be defined by a preference strategy. We will use Def. 1 for this purpose throughout this article. The second language, called `PDDL2.1`*, arises from a somewhat different context. One motivation has been to condense preference descriptions and the definition of the RKB into a single language. It is also closely related to the language of preferential algebras (Andreka, Ryan, & Schobbens 2002), our constructs `LEX` and `CAR` corresponding to the 'but' and 'on the other hand' operators introduced in the cited work. Furthermore PDDL2.1* offers new ways of combining numerical and qualitative measures.

An extension closely related to PDDL2.1^q, called PDDL3, was recently proposed by Gerevini and Long (Gerevini & Long 2005). Their extension differs from ours in several important aspects:

1. PDDL3 contains modal constructs which allow properties of plan trajectories to be expressed. For instance, the goal $(sometime\ g)$ expresses that at some intermediate state reached by the plan g should hold. We focus entirely on goal states.
2. Preferences in PDDL3 are numerical rather than qualitative. The plan representation may contain soft constraints. Violation of soft constraints induces certain costs. We want to be able to specify purely *qualitative* preferences in our PDDL language.

Of course, our proposed extensions can also be included in PDDL3.

The definition of PDDL2.1^q is a s follows:

```
<psp>        ::=(define (pspname <name>)
                        (:problem <name>)
                        [(:domain <name>)]
                        <goal>
                        <psp-def>)
<psp-def>    ::= (:psp <psp-node>+)
<psp-node>   ::= (<GD> positive integer)
```

Here <GD> is a goal descriptor. The integers following goal descriptors indicate the rank of the respective goal. We can thus represent RKBs in this language. The induced partial preorder on states corresponds to the one defined in Def. 1. The symbol <goal> permits the specification of obligatory goals and overwrites a corresponding entry in the problem description file.

The definition of PDDL2.1* differs from the one above in the description of <psp-def>:

```
<psp-def>    ::= (:psp <psp-node>)
<psp-node>   ::= (LEX <psp-node>+)
<psp-node>   ::= (CAR <psp-node>+)
<psp-node>   ::= (MULT <psp-node>+)
<psp-node>   ::= <GD>
<psp-node>   ::= non-negative integer
```

Intuitively, LEX represents a lexicographic ordering based on the values of psp-nodes, where the nodes are written in order of increasing importance. CAR allows to sum up values. In the special case where all values are 1 or 0 we obtain the cardinality of nodes with value 1. The MULT construct makes it possible to multiply values.

The precise semantics of this language is determined by its corresponding function $\mathrm{val} : S \times \texttt{PDDL2.1}^* \rightarrow \mathbb{N}$ which is given in table 1. For any PDDL2.1* expression n the preference relation $\succeq_n$ between to PDDL2.1 states s and s' is induced by the function val, i.e.

$$s \succeq_n s' \text{ iff } \mathrm{val}(s, n) \geq \mathrm{val}(s', n).$$

The parameters k_i in the expression for LEX are chosen such that higher values for nodes with higher index are preferred. This can be achieved by letting $k_i = 1 + maxval_{i-1}$ where $maxval_0 = 0$ and for $(1 \leq j \leq n)$ $maxval_j$ is the maximal value that can be obtained by the sum $\sum_{i=1}^{j-1} k_i \,\mathrm{val}(s, n_i)$. The exact inductive definition is straightforward and thus omitted.

PDDL2.1* expression n	evaluation function $\mathrm{val}(s, n)$
$n = (\texttt{LEX}\ \ n_1 \ldots n_l)$	$\mathrm{val}(s, n) = \sum_{i=1}^{l} k_i \,\mathrm{val}(s, n_i)$
$n = (\texttt{CAR}\ \ n_1 \ldots n_l)$	$\mathrm{val}(s, n) = \sum_{i=1}^{l} \mathrm{val}(s, n_i)$
$n = (\texttt{MULT}\ \ n_1 \ldots n_l)$	$\mathrm{val}(s, n) = \prod_{i=1}^{l} \mathrm{val}(s, n_i)$
$n =$ *goal description*	$\mathrm{val}(s, n) = \begin{cases} 1 & \text{if } s \vDash n \\ 0 & \text{otherwise} \end{cases}$
$n = d,\ d \in \mathbb{N}$	$\mathrm{val}(s, n) = d$

Table 1: The evaluation function $\mathrm{val} : S \times \texttt{PDDL2.1}^* \rightarrow \mathbb{N}$ maps into the domain of natural numbers. The constants k_i in the LEX construct are chosen in such a way that it is always preferable to have a higher value for a node with higher index. They can be calculated in advance.

Following the general discussion in the section on computing optimal plans, we introduce a function ϕ defined as $\phi(s) = \mathrm{val}(s, n) + 1$ if $s \in S_G$ and $\phi(s) = 0$ otherwise. We will call the non-negative integer $\phi(s)$ of a state s the numerical preference value for that state.

Preference descriptions derived from RKBs can be easily represented in this language.

Proposition 6 *Let $K = (F_1, \ldots, F_n)$ be a RKB with ranked sets $F_i = \{f_i^1, \ldots, f_i^{m_i}\}$ of formulae. The linearized preorder corresponding to val_K is equivalent to the preorder defined by the PDDL2.1* expression*

$$(\texttt{LEX}\ \ (\texttt{CAR}\ \ f_1^1\ \ldots\ f_1^{m_1})\ \ldots\ (\texttt{CAR}\ \ f_n^1\ \ldots\ f_n^{m_n})).$$

Consequently, within PDDL2.1* the user can define preference descriptions both quantitatively and qualitatively.

Implementation

According to Algorithm 1, a PSP is solved by translating the original PDDL2.1 description and the specification of the preference relation (given in PDDL2.1*) into a (modified) PDDL2.1 problem. To specify the restricted set of goal states S'_G we make use of a numerical comparison. In detail, the translation of the PDDL2.1* problem consists of several steps:

1. For each goal description g_i in the preference expression n a numerical variable w_i is introduced. Substituting the variables in the PDDL2.1* expression n will be denoted by writing $n[\{w\}_i]$.
2. A mechanism has to be devised such that w_i is 1 if the planner considers a state in which g_i is satisfied, and otherwise w_i is 0. This can be achieved by a modification of the actions of the original PDDL2.1 problem.
3. A numerical comparison $(\phi[\{w\}_i] > q_C)$ is added to the goal list. Here $\phi[\{w\}_i]$ denotes the numerical expression which is obtained by applying the ϕ function symbolically to the expression $n[\{w\}_i]$. This numerical comparison restricts the number of goal states to those states s which have $\phi(s) > q_C$.

```
(define
(pspname PSP868)
(:problem depotprob7512)
(:goal
  ( and
    (on crate0  pallet2)
    (on crate3  pallet1)
  )
)
(:psp
  ( LEX
    (available hoist2)
    (clear crate0)
    (lifting hoist2 crate2)
  )
) )
```

Figure 1: A complete PSP description in PDDL2.1*. Two obligatory goals are specified in the `:goal` field and three soft constraints in `:psp`. Among the latter, the goal `(available hoist2)` has the lowest preference and `(lifting hoist2 crate2)` the highest.

4. The algorithm has to give back the value $q'_C = \phi(s)$ of the found solvable goal state s of the modified PDDL2.1 problem. This value is determined by numerically evaluating the expression $\phi[\{w\}_i]$ in state s.

In our implementation, Algorithm 1 is performed by a Perl-script. The original PSP problem is parsed in a first step (using `bison` and `flex`) and translated into PDDL2.1 by modifying the actions and adding a numerical constraint as indicated above. It should be noted that in our implementation we restrict goal descriptions to literals. Furthermore, we allow only binary MULT constructs with at least one component being a non-negative integer. As described in detail in (Feldmann 2005), there is no difficulty in actually extending this implementation to the general case. We stick to the restricted language since our translation scheme would otherwise lead to non-linear numerical expressions, which Metric-FF is currently not able to handle.

As an example, a very simple PDDL2.1* description is shown in Fig. 1. Fig. 2 presents excerpts of the modified domain and Fig. 3 excerpts of the problem file. In both cases, the modifications due to the translation of PDDL2.1* into PDDL2.1 are indicated by bold letters. A complete version of this example can be found on our website http://www.physik.uni-leipzig.de/∼feldmann/psp/.

After the translation into PDDL2.1 the planner Metric-FF is called. It has been modified slightly to return the preference value of the found solution. With the help of his value, the numerical constraint is updated and the planner is evoked again. If the return value of Metric-FF is not strictly better than its previous value, the algorithm terminates.

We see the following advantages of our implementation: (1) it is based on a simple and easily extendible syntax, (2) it allows for the description of both qualitative and quantitative preferences, (3) soft constraints (preferences) and hard constraints (required goals) are combined in a natural way.

```
(:functions
 (LOAD_LIMIT ?T - TRUCK )
 (CURRENT_LOAD ?T - TRUCK )
 (WEIGHT ?C - CRATE )
 (FUEL-COST )
 (PSP_V-1 )
 (PSP_V-2 )
 (PSP_V-3 )
 (PSP_START_METRIC )
 (PSP_MAX_METRIC )
) (:action UNLOAD
 :parameters (?X - HOIST ?Y - CRATE
              ?Z - TRUCK ?P - PLACE)
 :precondition  (and (AT ?X ?P) (AT ?Z ?P)
                (AVAILABLE ?X) (IN ?Y ?Z))
 :effect
  (and (not (IN ?Y ?Z))
   (not (AVAILABLE ?X))
   (LIFTING ?X ?Y)
   (decrease (CURRENT_LOAD ?Z)(WEIGHT ?Y))
    (when (and(= ?X HOIST2))
          (and(assign (PSP_V-1) 0)))
    (when (and(= ?X HOIST2)(= ?Y CRATE2))
          (and(assign (PSP_V-3) 1)))
  )
)
```

Figure 2: An excerpt of the translated PDDL domain. The modified pieces are shown in bold face. The introduced numerical variables PSP_V-i correspond to the i-th soft constraint in the psp decription.

```
(:init
 ...
 (= (PSP_V-1) 1)
 (= (PSP_V-2) 1)
 (= (PSP_V-3) 0)
 (= (PSP_START_METRIC) 1)
 (= (PSP_MAX_METRIC) 7)
) (:goal
 (and (ON CRATE0 PALLET2)
      (ON CRATE3 PALLET1)
      (<= (PSP_START_METRIC)
          (+ (*  1(PSP_V-1))
          (+ (*  2(PSP_V-2))
             (*  4(PSP_V-3)))))
 )
)
```

Figure 3: An excerpt of the translated PDDL problem file. The modified pieces are shown in bold face. In addition to the two obligatory goals the problem description now also contains a numerical comparison.

```
(:goal ( and
 (( on crate0  pallet2  ) hard 1000 )
 (( on crate3 pallet1  ) hard 1000 )
 (( available hoist2 ) soft 2)
 (( clear crate0 ) soft 3)
 (( lifting hoist2 crate2 ) soft 5) ))
```

Figure 4: Excerpt of a `Sapa`^{ps} problem file constructed by translating the corresponding PSP in figure 1. The weights are calculated following table 1 except that an additional constant of 1 is added to each soft constraint.

Evaluation

In this section we present initial tests of our algorithm. For this purpose we use the depot domain enriched with numerical expressions which was a planning domain of the IPC3 competition. In order to check validity and correctness of our approach, we compare our results with `Sapa`^{ps} (Do & Kambhampati 2003), which is an existing planning system. The test runs are performed as follows. In a first step the domain and problem descriptions are parsed. This information is used to build our benchmark problems in a random fashion, i.e. rules of the `PDDL2.1`* language and the grounding instances of predicates are chosen randomly from a uniform distribution. The nesting level of our generated PSP ranges from 1 to 3. Literals are not allowed as top-level PSP-nodes in order to avoid an excessive number of problems containing just a single formula. The number of childnodes is restricted to 4 or less. For the sake of a clear comparison between our algorithm and `Sapa`^{ps}, we are neither invoking the `MULT` rule nor do we use negated literals. Between 1 and 31 soft constraints are contained in each of our benchmark problems. This distribution has an approximately normal shape with a mean of 12.9 and a standard deviation of 5.9 soft constraints. Additionally, a random selection of goals of the original `PDDL` problem description is transferred to our PSPs constituting hard (obligatory) goals.

In order to employ `Sapa`^{ps}, we transformed our `PDDL2.1`* problems into quantitive problems using weights on goals. This step is analogous to the translation of `PDDL2.1`* descriptions into the `PDDL2.1` language described in the last section. One particular example of the generated PSP problems is listed in Fig. 1. In Fig. 4 an excerpt from the corresponding `Sapa`^{ps} problem file is shown.

We would like to stress that `Sapa`^{ps} is a system designed for best-benefit problems which differ from our problems in taking costs of actions into account. The main focus of `Sapa`^{ps} lies on solving quantitative problems while our algorithm can also deal with purely qualitative problems. On the other hand, `Sapa`^{ps} can handle time domains which our implementation so far cannot.

It should also be noted that `Sapa`^{ps} cannot parse numerical comparisons. Since the original depot domain contains such a comparison, we use for both planners a modified version of the original depot domain where the numerical comparison is removed. When considering and comparing running times it should be stressed that `Sapa`^{ps} is a Java-based application while our algorithm employs

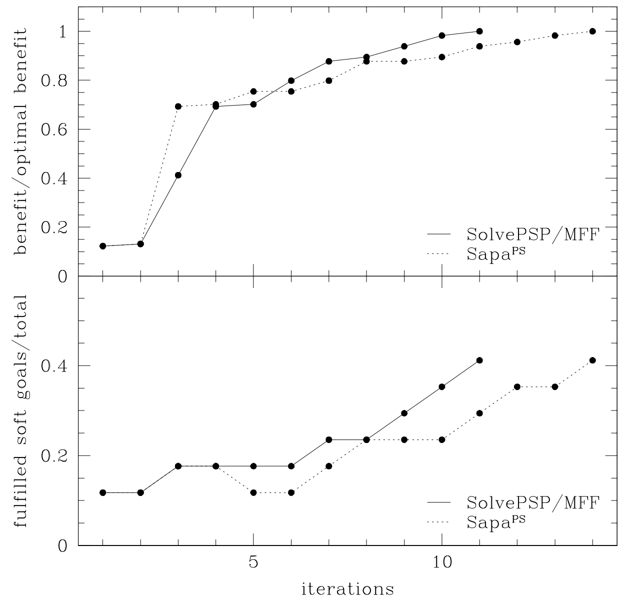

Figure 5: The results for a particular problem are presented which contains no hard goals and 17 soft goals. In this example, both our approach (solid line) and `Sapa`^{ps} (dotted line) could find a satisfiable goal state. The number of iterations refers to either the number of loop passages of our algorithm or to the number of internal cycles of `Sapa`^{ps}. Top: the development of the benefit normalized to the maximal possible benefit. Bottom: The number of satisfied soft constraints normalized to their total number. The search times for this example are 100ms (our algorithm) and 23s (`Sapa`^{ps}), respectively.

`Metric-FF` which is written in C.

The approach of `Sapa`^{ps} relies on action costs to guide its search. Therefore, assigning non-zero action costs is essential in order to compare the two systems. More precisely, too small (<0.01) or too high (>100) action costs prevent `Sapa`^{ps} from finding a solvable goal state when run on our benchmark problems. We found that cost values of about 0.1 seem to work best. The action costs are re-added to the benefit after each run. In conclusion, we emphasize that `Sapa`^{ps} might not be the optimal choice for comparison purposes but it is one of the best systems at hand at the moment.

All tests were performed on a AMD Athlon 64 4000+ with 2GB RAM running Debian Linux. We generated 1000 benchmark problems, characterized by different numbers of hard and soft constraints. A maximal heap space of 1GB was allowed for the Java machine running `Sapa`^{ps}. This value was chosen after initial tests and seems to be a good choice. In order to obtain results in manageable time, we assigned timeouts to both systems. For the `Sapa`^{ps} system we chose a time limit of 60s for each problem. For our system we tried several time limits ranging from 100ms to 60s.

Fig. 5 shows a particular problem which both planners were able to solve. This problem contains no hard and 17

problems	posed	plan found			better benefit	
		MFF	sapa	both	MFF	sapa
Total	1000	994	507	504	336	1
HG0_SG00-05	24	24	24	24	7	-
HG0_SG05-10	70	70	70	70	42	-
HG0_SG10-15	90	90	90	90	65	-
HG0_SG15-20	85	85	84	84	71	-
HG0_SG20-50	44	41	42	39	33	1
HG1_SG00-05	33	33	13	13	2	-
HG1_SG05-10	87	87	40	40	18	-
HG1_SG10-15	136	136	59	59	44	-
HG1_SG15-20	90	90	31	31	28	-
HG1_SG20-50	69	67	14	14	10	-
HG2_SG00-05	12	12	3	3	1	-
HG2_SG05-10	59	59	16	16	3	-
HG2_SG10-15	58	58	9	9	3	-
HG2_SG15-20	64	64	8	8	8	-
HG2_SG20-50	25	24	1	1	1	-
HG3_SG00-05	10	10	2	2	-	-
HG3_SG05-10	5	5	1	1	-	-
HG3_SG10-15	17	17	-	-	-	-
HG3_SG15-20	14	14	-	-	-	-
HG3_SG20-50	5	5	-	-	-	-
HG4_SG05-10	1	1	-	-	-	-
HG4_SG15-20	2	2	-	-	-	-

Table 2: The statistical results comparing our algorithm and `Sapa`^{ps} on the generated benchmark problems. A time limit of 10s has been applied to our algorithm. For the `Sapa`^{ps} system we chose a larger time limit of 60s. The partial satisfaction problems are classified according to the number of soft and hard goals, i.e. HGi_SGj-k corresponds to the set of problems containing i hard goals and between j (inclusive) and k (exclusive) soft constraints. The number of problems in set HGi_SGj-k is given in the second column. The third column shows the number of problems for which a solvable goal state and a corresponding plan has been found. The last column compares the benefit of the problems for which both planners found a solvable goal state.

soft goals. Evidently, the behavior is rather similar for a low number of iterations, but our system converges faster with respect to the number of iterations and with respect to searching time. In this example, both approaches find an optimal solution of the PSP. The optimality of the solution could be confirmed by our algorithm setting the time limit to 60s. This specific example is not representative, but demonstrates the proper working of our algorithm.

The results of both planners on our benchmark problems are summarized in Tab. 2. We note that for almost all problems our algorithm could find at least a satisfiable goal state. Interestingly, some of the unsolved problems have a trivial solution. Regarding the `Metric-FF` system this is due to the fact that only up to 25 soft constraints can be specified. We could have increased this limit by adapting the planner's source code. Following our methodology of being planner independent, we decided to avoid such modifications. We added, however, a simple output function to return the reached numerical preference value. In Tab. 2 the difficulty of problems increases from top to bottom. At the same time the number of problems for which `Sapa`^{ps} can find a solvable goal state decreases. Choosing a larger time limit might help to obtain better results for `Sapa`^{ps}. Our approach performs well on all our benchmark problems. In a future analysis it would be interesting to evaluate our algorithm on even more difficult problems.

time limit	plan found	optimal plan found	optimality proved
0.1s	994	640	47
1s	994	892	49
10s	994	988	186
60s	994	989	989

Table 3: The optimality results of our approach using various time limits from 100ms to 60s. Smaller limits have not been used due to the low precision of time measurements below 100ms. The second column shows the number of problems for which a solvable goal state was found. The third column states the number of PSPs for which an optimal solution was computed within the time limit, independently of whether optimality was proven by the system. The last column indicates for how many problems optimality could actually be proven within the time limit.

So far we have addressed the question of whether a solvable goal state can be found. In order to actually solve a partial satisfaction problem, we need to find an optimal solvable goal state. In fact, we want to distinguish two different optimality results. A planner can find a solution of a given PSP with and without knowing that the found solvable goal state is actually optimal. In order to prove optimality, the planner has to perform an unsuccessful and thus time-consuming search. Considering this difference, the results of our algorithm on the benchmark problems are shown in Tab. 3. For most of the benchmark problems our algorithm finds an optimal solvable goal state very quickly, i.e. within a fraction of a second. We also confirm that it takes a rather long time to prove optimality of the found solution.

In summary, we have demonstrated that our approach of solving PSPs actually works in practical situations, is flexible and applicable to a large range of different problem sizes. For the benchmark problems at hand, our approach generally returns a plan of equal or better quality than the `Sapa`^{ps} system. In addition, since we use `Metric-FF` as underlying planner, our approach is much faster than `Sapa`^{ps}.

Testing plan optimality

So far we have discussed in this paper the following question:

> Given a PSP Γ, specified via a ranked goal base K, how do we compute an optimal plan for Γ?

The answer we gave was based on a strengthening of the partial order on plans induced by K. The idea was to come up with a linearization whose optimal elements are guaranteed to be optimal elements of the original partial ordering on plans. This linearization can easily be translated to a numerical measure, which is then represented in a corresponding metric to be used incrementally as a lower bound for the computation of plans.

In this section we want to address the following question:

Given a PSP Γ, specified via a ranked goal base K, and a plan P. How do we check whether P is optimal?

Note that this is not as trivial as it may appear at first sight. We cannot just compute the metric we used so far, compute the value for plan P and check whether a plan with higher value exists. The reason is that, although we are guaranteed to find an optimal plan this way, it is not guaranteed that an optimal plan is also optimal with respect to the linearization. Indeed, in most cases some optimal plans will become suboptimal after linearization.

Consider a simple example. We have the following RKB of goals:

$$(\{d\}, \{a, b, c\})$$

that is, a, b and c are the most important goals, d is less important than the others. Assume there are 2 plans achieving maximal subsets of all goals, P_1 with goal state $s_1 = \{a, b\}$ and P_2 with goal state $s_2 = \{c, d\}$. P_1 and P_2 are incomparable since their sets of reached goals of highest importance, $\{a, b\}$ and $\{c\}$, respectively, are not in subset relation. However, the metric we used so far strictly prefers P_1. Simply using this metric and checking whether a plan with higher value exists thus does not give the correct results.

For testing optimality of plans another metric is needed. This metric will have to depend on the plan P to be tested. The idea is to use a metric where the satisfaction of a new goal of the same level can only lead to a higher overall value whenever all goals of this level satisfied by P are also satisfied.

The comparison of plans is based on the comparison of the goals they achieve. For this reason it is sufficient to define the metric on sets of ranked goals. Let $K = (G_1, \ldots, G_n)$ be the given RKB of goals, i.e. elements of G_n are of highest priority, those of G_{n-1} of second highest etc. Let $(S_1, \ldots, S_n)$ be the sequence of subsets $S_i \subseteq G_i$ of goals satisfied by plan P, $(N_1, \ldots, N_n)$ those goals not satisfied by P (i.e., $N_i = G_i \setminus S_i$).

We need to find a numerical measure val_P assigning an integer to an RKB R such that $val_P(R) > val_P(S_1, \ldots, S_n)$ iff R is strictly preferred to $(S_1, \ldots, S_n)$. Now if there is a plan obtaining goals with measure higher than that of P, then P is not optimal.

We define val_P inductively as follows (in principle, all functions should have an additional index P. We omit this index here for readability):

$$maxval_0 = 0$$

for each j ($1 \leq j \leq n$) let

$$\begin{aligned} negval_j &= maxval_{j-1} + 1 \\ posval_j &= |N_j| \times negval_j + maxval_{j-1} + 1 \\ maxval_j &= |S_j| \times posval_j + |N_j| \times negval_j \\ &\quad + maxval_{j-1} \end{aligned}$$

Let $R = (R_1, \ldots, R_n)$ be an RKB. We define

$$val_P(R) := \sum_{i=1}^{n} (|R_i \cap S_i| \times posval_i + |R_i \cap N_i| \times negval_i).$$

Intuitively, $negval_j$ is the value of an unsatisfied goal from level j, $posval_j$ is the value of a satisfied goal from that level, $maxval_j$ is the maximal value which can be obtained by satisfying all goals of level j and below, and $val_P(R)$ is the sum of all values of goals in any level of R. The values are chosen such that satisfying an additional goal at level j leads to a higher overall value provided the same goals are satisfied at all levels with higher index. We will also use the notation $val_P(g)$ for a goal g with the obvious meaning: $val_P(g) = posval_j$ iff $g \in S_j$ and $val_P(g) = negval_j$ iff $g \in N_j$.

Proposition 7 *Let $(S_1, \ldots, S_n)$ be the goals satisfied by plan P. P is an optimal plan iff there is no plan P' such that*

$$val_P(S_1, \ldots, S_n) < val_P(S'_1, \ldots, S'_n),$$

where S'_j are the goals contained in G_j satisfied by P'.

Proof: Assume P is not optimal. Then there is a plan P' and level k such that P and P' satisfy the same goals in levels higher than k, and P' satisfies a proper superset of goals in level k. Since by construction satisfying an additional goal in level k adds a higher value than satisfying an arbitrary set of goals from levels $1, \ldots, k-1$, the overall value for P' is higher than that for P.

Similarly, if P is optimal then we have for each plan P' that either P' satisfies exactly the same goals as P (in which case the overall value is not higher than for P), or there is a level k such that P and P' satisfy the same goals in all levels $j > k$, and P' does not satisfy some goal g of level k satisfied by P. Again, by construction, the loss by not satisfying g is higher than the maximal gain obtained by satisfying any goal of level $i \leq k$ not satisfied by P. The overall value for P' is thus not higher than that for P. $\Box$

Consider our example. Since P_2 achieves $(\{c\}, \{d\})$, we obtain the following values:

$$val_{P_2}(d) = 1 \quad val_{P_2}(a) = 2$$
$$val_{P_2}(b) = 2 \quad val_{P_2}(c) = 6$$

The overall value for P_2 is thus 7, the value for P_1 is 4, P_2 is thus optimal. Of course, we also establish that P_1 is optimal. It is easy to verify that in this case the metric yields

$$val_{P_1}(d) = 1 \quad val_{P_1}(a) = 4$$
$$val_{P_1}(b) = 4 \quad val_{P_1}(c) = 2$$

With these values we obtain an overall value of 8 for P_2, a value of 3 for P_1. We have established that P_2 is an optimal plan as well.

With these results we can compute the optimality test for P as follows: we first generate $val_P(g)$ for each goal and compute the overall value for P. We then add the description of the metric to the plan description using the overall value, incremented by 1, as lower bound. P is optimal iff no plan satisfying this bound exists.

Discussion and related work

(Brafman & Chernyavsky 2005) present an approach to planning with goal preferences which shares a lot of motivation with our proposal, but which also differs in several important aspects. Their work is based on a particular planning method developed in (Do & Kambhampati 2001). The planning problem is converted into an equivalent constraint satisfaction problem (CSP) using a Graphplan encoding (Blum & Furst 1997). Brafman and Chernyavsky then use an algorithm for constrained optimization over CSPs (Brafman & Domshlak 2002) which uses so-called TCP-nets, a generalization of CP-nets, for preference elicitation and representation.

Firstly, our proposal differs from this approach in the way preferences are represented. Rather than CP-nets which give preferences a *ceteris paribus* (other things being equal) interpretation under which only states differing in exactly one atom can be directly compared, we use ranked knowledge bases representing ranked user goals. As demonstrated in (Coste-Marquis *et al.* 2004), CP-nets cannot represent arbitrary preferences among states, a restriction which does not apply to RKBs. Secondly, whereas the approach in (Brafman & Chernyavsky 2005) depends on a particular planning method, our approach is independent of the method chosen for classical planning and is thus able to benefit from further developments in this area. All we require from the classical planner is its ability to handle numerical values adequately.

There are also several related papers on oversubscription planning (van den Briel, Nigenda, & Kambhampati 2004; Smith 2004), i.e. planning with a large number of goals which cannot all be achieved. The major difference here is that we use qualitative preferences whereas the cited papers add real-valued weights to the goals which then are used for computing preferences. In many settings qualitative preferences appear more natural and are easier to elicit from users than numerical preferences. Moreover, as pointed out in (Brafman & Chernyavsky 2005), the algorithms used in both papers are not guaranteed to reach an optimal solution.

In (Eiter *et al.* 2002) an answer set programming approach to planning under action costs is presented. Here the criterion for plan optimality is not the quality of the reached goal state, but the accumulated costs of actions in the plan.

Son and Pontelli (Son & Pontelli 2004) define a flexible language for expressing qualitative preferences in answer set planning. The language includes temporal logic constructs for expressing preferences among trajectories. We focus here on preferences among goal states in the context of classical planning approaches.

The authors of (Delgrande, Schaub, & Tompits 2004) introduce two types of preferences among trajectories of transition systems, choice preferences and temporal preferences. They later show that the latter actually can be reduced to the former. As in our approach, formulae are used to express preferences, but there are no preferences among the formulae themselves. Furthermore, the mentioned authors are interested in the question whether a formula is satisfied somewhere in the history, whereas we consider the satisfiability in the final state only. Moreover, computational aspects play a minor role in (Delgrande, Schaub, & Tompits 2004).

In this paper we presented an approach to prioritized planning which uses RKBs to express qualitative preferences among goal states. To compute an optimal plan we translate the preference preorder on states to a valuation function such that optimal states with respect to the latter are guaranteed to be optimal with respect to the former. A similar translation method allows us to test plans for optimality. Our algorithm computes a sequence of strictly improving plans and is guaranteed to terminate with an optimal plan. Furthermore, our implementation is independent of a particular approach to classical planning which we see as an important advantage of our proposal. Results of an empirical evaluation we presented are promising.

Acknowledgements

We would like to thank the developers of $\texttt{Sapa}^{ps}$, in particular J. Benton and M. van den Briel, for their help and support during the evaluation and comparison of our systems.

References

Andreka, H.; Ryan, M.; and Schobbens, P. 2002. Operators and laws for combining preference relations. *JLC: Journal of Logic and Computation* 12.

Benferhat, S.; Cayrol, C.; Dubois, D.; Lang, J.; and Prade, H. 1993. Inconsistency management and prioritized syntax-based entailment. In *Proc. IJCAI-93*, 640–647.

Blum, A., and Furst, M. L. 1997. Fast planning through planning graph analysis. *Artif. Intell.* 90(1-2):281–300.

Brafman, R. I., and Chernyavsky, Y. 2005. Planning with goal preferences and constraints. In Biundo, S.; Myers, K. L.; and Rajan, K., eds., *Proc. ICAPS-05*, 182–191. AAAI.

Brafman, R. I., and Domshlak, C. 2002. Introducing variable importance tradeoffs into CP-nets. In Darwiche, A., and Friedman, N., eds., *Proc. UAI-02*, 69–76. Morgan Kaufmann.

Brewka, G. 1989. Preferred subtheories: An extended logical framework for default reasoning. In *Proc. IJCAI-89*, 1043–1048.

Brewka, G. 2004. A rank based description language for qualitative preferences. In de Mántaras, R. L., and Saitta, L., eds., *Proc. ECAI-04*, 303–307. IOS Press.

Coste-Marquis, S.; Lang, J.; Liberatore, P.; and Marquis, P. 2004. Expressive power and succinctness of propositional languages for preference representation. In *Proc. KR-04*, 203–212.

Delgrande, J. P.; Schaub, T.; and Tompits, H. 2004. Domain-specific preferences for causal reasoning and planning. In *Proc. KR-04*, 673–682.

Do, M. B., and Kambhampati, S. 2001. Planning as constraint satisfaction: Solving the planning graph by compiling it into CSP. *Artif. Intell.* 132(2):151–182.

Do, M. B., and Kambhampati, S. 2003. Sapa: A multi-objective metric temporal planner. *J. Artif. Intell. Res. (JAIR)* 20:155–194.

Dubois, D.; Welty, C. A.; and Williams, M.-A., eds. 2004. *Principles of Knowledge Representation and Reasoning: Proceedings of the Ninth International Conference (KR2004), Whistler, Canada, June 2-5, 2004*. AAAI Press.

Eiter, T.; Faber, W.; Leone, N.; Pfeifer, G.; and Polleres, A. 2002. Answer set planning under action costs. In Flesca, S.; Greco, S.; Leone, N.; and Ianni, G., eds., *Proc. JELIA-02*, volume 2424 of *Lecture Notes in Computer Science*, 186–197. Springer.

Feldmann, R. 2005. *Planen mit Präferenzen - Ein Ansatz zur Lösung partieller Erfüllbarkeitsprobleme*. diploma thesis, Leipzig University.

Fikes, R., and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artif. Intell.* 2(3/4):189–208.

Fox, M., and Long, D. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *J. Artif. Intell. Res. (JAIR)* 20:61–124.

Gerevini, A., and Long, D. 2005. Plan constraints and preferences in PDDL3, AIPS-06 planning committee.

Ghallab, M.; Howe, A.; Knoblock, C.; McDermott, D.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL—the planning domain definition language, AIPS-98 planning committee.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. Morgan Kaufmann.

Goldszmidt, M., and Pearl, J. 1991. System-Z+: A formalism for reasoning with variable-strength defaults. In *Proc. AAAI-91*, 399–404.

Hoffmann, J. 2003. The Metric-FF planning system: Translating ”ignoring delete lists” to numeric state variables. *J. Artif. Intell. Res. (JAIR)* 20:291–341.

Pearl, J. 1990. System Z: A natural ordering of defaults with tractable applications to nonmonotonic reasoning. In Parikh, R., ed., *Proc. TARK-90*, 121–135. Morgan Kaufmann.

Pednault, E. P. D. 1989. ADL: Exploring the middle ground between STRIPS and the situation calculus. In *Proc. KR-89*, 324–332.

Smith, D. E. 2004. Choosing objectives in over-subscription planning. In Zilberstein, S.; Koehler, J.; and Koenig, S., eds., *Proc. ICAPS-04*, 393–401. AAAI.

Son, T. C., and Pontelli, E. 2004. Planning with preferences using logic programming. In Lifschitz, V., and Niemelä, I., eds., *Proc. LPNMR-04*, volume 2923 of *Lecture Notes in Computer Science*, 247–260. Springer.

van den Briel, M.; Nigenda, R. S.; Do, M. B.; and Kambhampati, S. 2004. Effective approaches for partial satisfaction (over-subscription) planning. In McGuinness, D. L., and Ferguson, G., eds., *Proc. AAAI-04*, 562–569. AAAI Press / The MIT Press.

van den Briel, M.; Nigenda, R. S.; and Kambhampati, S. 2004. Over-subscription in planning: a partial satisfaction problem. In *ICAPS-04 Workshop on Integrating Planning into Scheduling*.